Improving Throughput in SCTP via Dynamic Optimization of Retransmission Bounds

Eduardo Gonzalez

FlexRadio Systems

4616 W. Howard Lane Suite 1-150

Austin, TX USA 78728

Email: ed.gonzalez@flex-radio.com

Wuxu Peng

Texas State University

601 University Drive

San Marcos, TX USA 78666

Email: wuxu@txstate.edu

Stan McClellan

Texas State University

601 University Drive

San Marcos, TX USA 78666

Email: stan.mcclellan@txstate.edu

Received: August 30, 2015 Accepted: Nov. 1, 2015 Published: Nov. 28, 2015

DOI:10.5296/npa.v7i3.8228 URL: http://dx.doi.org/10.5296/npa.v7i3.8228

Abstract

The Stream Control Transmission Protocol (SCTP) is a relatively new transport protocol. It has several underlying mechanisms that are similar to the Transmission Control Protocol (TCP), as well as several improvements that are important in certain classes of applications. The timeout scheme of SCTP, however, is almost identical to that used in TCP. With the dynamics of today's Internet, that timeout scheme may be too passive. This paper presents an algorithm which dynamically adjusts the overall context of the retransmission timeout process without changing the fundamental retransmission mechanisms. This approach manages the impact of fast retransmissions and timeouts to significantly improve the throughput of SCTP applications. The algorithm has been implemented and tested in real network environments. Experimental results show that the algorithm avoids spurious retransmissions and provides better throughput by intelligently managing RTO (retransmission timeout) boundaries and allowing conventional timeout schemes to participate more actively in the retransmission process.

Keywords: SCTP, TCP, RTT, RTO, Jacobson algorithm

1 Introduction

As a relatively new transport layer protocol, SCTP provides some salient features which are absent from UDP and TCP: message bundling; multi-streaming; multi-homing; built-in reachability checking; and state cookie support [1–3]. As a connection oriented protocol, the timeout mechanism in SCTP is critical to its performance. The foundation of SCTP's timeout algorithm is the well known Jacobson algorithm [4–6] which has long been an important cornerstone of TCP. Jacobson's algorithm is simple, effective, and well-supported by fundamental theory. Performance issues related to Jacobson's algorithm and other retransmission procedures have been noted and addressed in several alternative approaches [7–9].

Unfortunately, in many cases we note that Jacobson is prevented from functioning effectively due to implementation of the minimum value for RTO ("RTOmin") which is defined as 1 second for SCTP [2]. With low RTT (round trip time) values characteristic of modern networks, optimum RTO values such as those produced by Jacobson are typically much lower than RTOmin. Thus, although Jacobson's algorithm is always used to *compute* the optimum retransmission timeout, the RTO value that is *actually used* is RTOmin. Unfortunately, simply lowering the static value of RTOmin can cause problems with spurious retransmissions as network conditions change. What is needed is a dynamic approach rather than a simple, fixed threshold.

This paper presents a bounding approach for RTOmin that extends the classic Jacobson's algorithm and replaces the static value of RTOmin with a dynamic threshold. The new algorithm takes into account important factors – fast retransmission, timeout, and history of timeout – that impact transmission performance of SCTP. The algorithm has been implemented as a Linux kernel module and tested in multiple real network environments. We show that, when coupled with Jacobson's algorithm, the new RTO approach intelligently manages the retransmission process, dynamically adapts to current network conditions, and significantly improves the throughput of SCTP applications as measured by goodput. Since the algorithm focuses on optimizing the *RTOmin boundary* rather than the *optimal RTO estimate*, it also maintains stability without causing unnecessary fast retransmissions. The concept is validated by network testing which shows that the dynamic algorithm intelligently balances between fast retransmissions and timeouts, enhancing throughput while maintaining network stability.

The classical timeout scheme used in TCP and subsequently in SCTP is simple and effective and has played important roles of stabilizing Internet [1, 2, 4–6]. But with ever changing and evolving dynamics of today's Internet, newer and more actively managed timeout schemes are needed. Our approach is based on the belief that there must be a fine balance that can be exploited to both avoid unwanted characteristics and benefit from more proactively managed retransmission behavior. Our algorithm can be easily integrated into current SCTP implementations. It can provide desired performance enhancement with no need of change to the established network structure.

The rest of this paper is organized as follows: Section 2 briefly reviews related work and contrasts our research with previous approaches. Section 3 examines several critical aspects of the existing timeout/retransmission algorithm used in SCTP. In Section 4 we introduce and discuss a test suite that was created for effective and efficient testing of new network parameters and algorithms. The test suite provides three real-time network environments which can be tailored to test different aspects of transport layer protocols. In Section 5 a modified SCTP RTOmin algorithm that manually modifies the RTOmin values is introduced. We observe from the experimental results of the manual RTOmin algorithm that there are opportunities selecting an optimal or locally optimal RTOmin value for any specific network environment. Section 6 introduces our dynamic RTOmin algorithm and its implementation in Linux kernel. The proposed algorithm dynamically adjusts RTOmin values in SCTP stack based on the ratio of fast retransmissions and timeouts. We argue that this ratio allows the proposed algorithm to differentiate congestion and network loss and hence intelligently manage change of network dynamics. Section 7 describes the approaches used for testing the dynamic RTOmin algorithm, as well as the performance of the algorithm in real network environments. Section 8 provides concluding remarks and ideas for future work.

2 Related work

Significant effort has been devoted to the study of the performance and possible enhancement of retransmission mechanisms in IP transport protocols, such as mobile adhoc networks (MANET) [10]. Here, we discuss a limited set of work specifically related to the dynamic RTOmin algorithm.

One general approach to managing retransmissions lies in discriminating between *thick* and *thin* streams and applying separate retransmission rules depending on the classification [11]. A thick stream attempts to fully saturate available bandwidth to maximize goodput while a thin stream is characterized by small, widely spaced packets that require little bandwidth, but are often time critical. Most enhancements in [11] are highly invasive, modifying fast retransmit and timeout behaviors and firing the fast retransmission mechanism after a single SACK (selective acknowledgment) that reports missing chunks. Other work [12] proposes similar mechanisms. In contrast, the dynamic RTOmin algorithm makes a subtle but important change to the (previously static) value of RTOmin, allowing intelligent management of stream context without invasive modifications. Thus, the new algorithm is applicable to all types of streams and has beneficial outcomes regardless of the way the application may be using the protocol.

Some approaches advocate lowering the value of retransmission timers to improve transmission performance [13, 14]. This has the undesirable side-effect of uncontrolled, spurious retransmissions, too many selective acknowledgments, and and conflicts with other mechanisms. In contrast, the dynamic RTOmin algorithm provides evidence in favor of lower bounds but only in select environments which are automatically detected.

Our approach is based on the belief that intelligent management of retransmission can avoid unwanted characteristics while benefiting from more aggressive retransmission behavior.

The work in [15] replaces well-known retransmission mechanisms in an attempt to better suit the application. In contrast, our work leverages existing algorithms and balances their activities intelligently. Rather than requiring invasive modifications or replacement of established processes, the dynamic RTOmin algorithm can be seamlessly integrated into current SCTP implementations. It can provide desired performance enhancement with minimal change to the established network structure.

In [16] the authors argued and showed through experiments that RTOmin can be dynamically adjusted to affect SCTP's throughput. The paper showed that there are opportunities to adjust RTOmin values to provide globally or locally optimal goodput. However that work is one step of short of how to integrate RTOmin adjustment into current SCTP's timeout mechanism. More about this will be discussed later in this paper.

In an interesting paper [17] Chowdbury and Jony shows that an SCTP node in NS2 cannot be truly multi-homed. Hence, NS2 still lacks of full implementation of SCTP features.

More recently Najm and et al. proposed to use SCTP to relieve congestion control problem in LTE-A [18].

3 Retransmission mechanisms

This section briefly describes three core concepts of SCTP retransmission mechanisms – *fast retransmissions*, *timeouts*, and *selective/delayed acknowledgments*. Details of these schemes can be found in the references [1–3, 10, 19].

Selective Acknowledgements (SACK) were initially introduced in [5] as a TCP option to handle multiple dropped packets within a window. This approach was later refined in [20] and adopted by SCTP. SACKs and selectively repeated retransmissions can be useful when an aggressive sender retransmits packets after multiple drops within a window. As in TCP, SCTP also employs *Delayed Acknowledgments* (DACK). An SCTP receiver typically waits to ACK an individual chunk in the hope of piggybacking several ACKs with a subsequent reverse-path data chunk. In most Linux kernel implementations, the DACK/SACK delay is around 200 ms.

Fast retransmissions provide congestion avoidance and lost data corrections by selectively retransmitting SCTP chunks. If four acknowledgments report the same missing chunks, those chunks are re-sent in the next available packet. Unfortunately, fast retransmission is invoked only after multiple indications of the same missing chunks, not after a time threshold is crossed. As a result, fast retransmission mechanisms do not distinguish between network loss/corruption and congestion, responding identically in both cases [10, 21].

Timeout mechanisms, including Jacobson's algorithm, are focused primarily on link failure detection, as well as congestion mitigation. We have previously discussed the theoretical basis for Jacobson [9]. Unfortunately, in practical implementations *the optimum*

RTO value computed by Jacobson is typically discarded. If the Jacobson calculation is outside of the window bounded below by RTOmin (1000 msec) and above by RTOmax (60 sec), the optimal RTO value is replaced with the bounding value. With low RTT values characteristic of modern networks, optimum RTO values are typically lower than the RTOmin boundary. Thus, although Jacobson's algorithm is always used to *compute* the optimum retransmission timeout, the RTO value that is *actually used* is the static RTOmin boundary of 1000 msec.

In summary, the standard value of RTOmin (1000 msec) causes the timeout mechanism to remain dormant in modern, fast networks until the entire link is down. In other words, the Jacobson algorithm is largely prevented from affecting the timeout process due to RTOmin. This leaves the fast retransmission and DACK/SACK mechanisms responsible for detecting network congestion *as well as* quickly recovering lost packets.

4 Test Environments

In this section we briefly introduce a suite of test environments that we created and used to validate the proposed research. The test suite consists of three real-time network environments [22]. Although all three environments were designed to support our research on SCTP, they can be readily modified to test other TCP/IP protocols such as TCP and UDP.

Typically research work like the one we are presenting is first verified with simulations. It's well known that simulations have several inherent limitations. We evaluated the performance of the dynamic RTOmin algorithm against the current "static algorithm" in a variety of real network environments. A driving factor for emphasizing real network performance comes from the observation that approximately 50% of SCTP research is done in simulation with limited use of real world scenarios [23], and the fact that NS2 simulations of SCTP may be lacking in several areas [17].

In all three environments implemented in this research work, a server host will wait for incoming connections which would be initiated by a client host. The SCTP protocol stacks of server hosts were not modified. Server hosts simply run logging software in order to capture performance statistical parameters of interests. Client hosts run Debian Linux as we found that Debian Linux is the only available Linux dialect that allows dynamic module loading/unloading. The client hosts would dynamically swap between unmodified and modified SCTP Kernel modules according to our control script. This set up was chosen over dedicating one client host executing modified SCTP protocol stack and another host running un-modified SCTP protocol stack since it reduces the likelihood of configuration and hardware discrepancy between client hosts. In addition to controlling swapping of SCTP protocol modules, the control script also controls various aspects of an experiment run such as size of data blocks to be transferred, SCTP chunk size, data loss rate, and etc. The control script controls network condition parameters such as loss and delay by manipulating the client NICs using the Network Emulator [24] linux package. The statistics of experimental runs of our new algorithms is compared with the existing Linux implementation of SCTP [25].

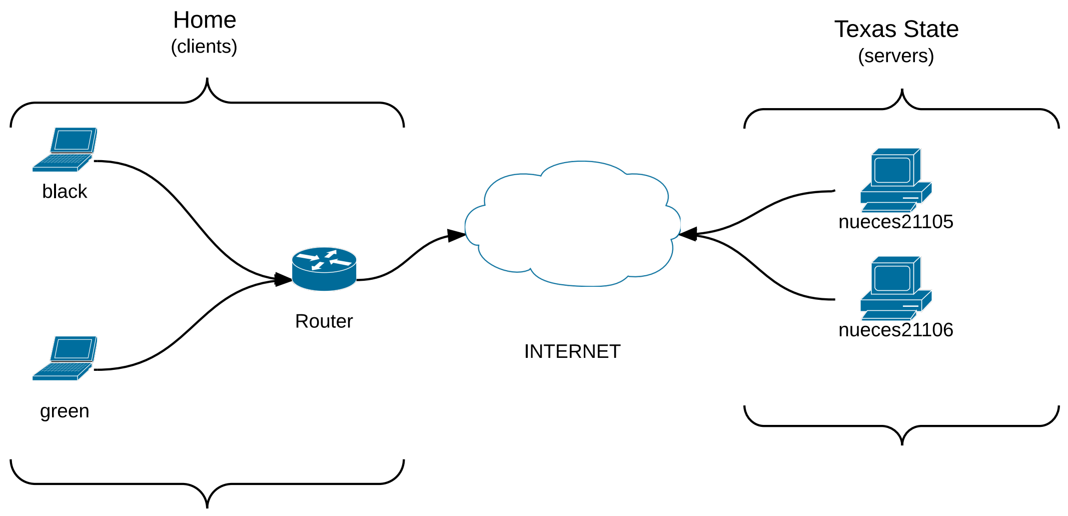

Figure 1: LAN Internet testing environment

4.1 Live-Internet Environment

The live-Internet environment, as shown in Figure 1, was used to test SCTP performance in medium latency networks with sporadic deviations. It can employ multiple clients. As shown in the figure there are two client hosts (black and green) which are Asus EEPCs running 32 bit Debian with linux kernel version 2.6.32. These two client hosts are connected to a router that has Internet service provided by a local ISP (Internet service provider) with a maximum downlink speed of 30 Mbps and a maximum upload speed of 5 Mbps. The clients communicate with two server hosts located within the Texas State University campus running 32 bit CentOS 6. To facilitate implementation and make uniform comparison both the server and client hosts run the same version of Linux version. The average RTT of this configuration was observed to be about 150 ms during the first phase of tests and 55 ms during the second phase.

4.2 LAN Local Environment

The LAN local environment was designed to capture behaviors of SCTP in low latency, high throughput scenarios. As illustrated in Figure 2 the environments consists of the two client hosts (named *black* and *green*) communicating with a server host (named *white*) located within the same LAN. As with the live-Internet environment the server is passive and waits for connection actions initiated by client hosts. The server logs relevant parameters while the clients would initialize connections, swap between modified and unmodified kernel modules and log relevant parameters of interest.

4.3 Virtualized Local Environment

The last testing environment, virtualized-local environment, is of particular interests. Typical TCP/IP protocols, including SCTP, were designed when virtual machines and networks were not proposed/popular yet. Given the increasing popularity and importance of virtual machines and networks, understanding behavior of TCP/IP protocols under virtual machines and networks is of special importance. Our virtualized-local environ-

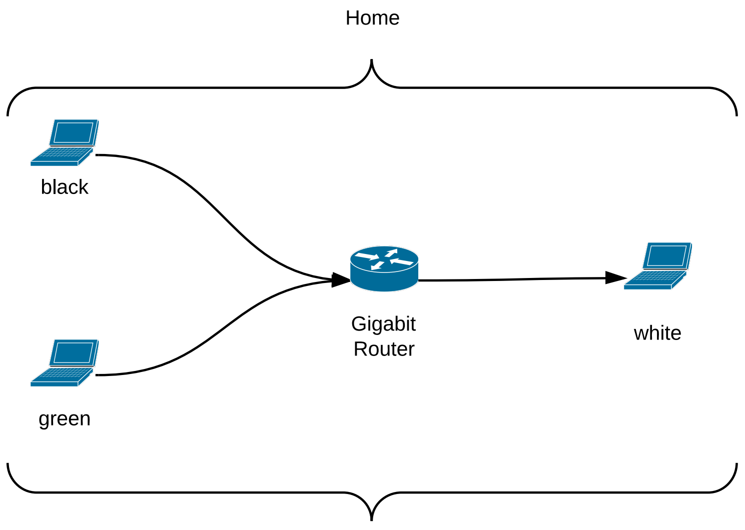

Figure 2: LAN local testing environment

ment provides a viable tool to test SCTP in virtual machine and network environments. The environment consists of completely virtualized network traffic as shown in Figure 3. This environment is ideal to test SCTP in a network with very low latency and high throughput such as inter-datacenter and intra-datacenter communications. Currently the environment consists of 3 virtual hosts, all running 64 bit Debian, Kernel version 2.6.32, within Texas State University's JCK datacenter. The client host (named *pronto*) communicates with server host (named *quick*) through a virtual bridge within the routing host (named *fast*). This configuration provides almost absolute control over network traffic and has an average RTT of 0-1 ms. As in previous two environments the SCTP kernel module will be dynamically swapped between unmodified and modified states within the client.

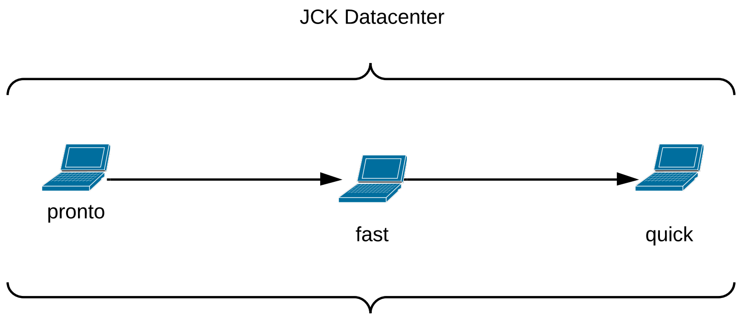

Figure 3: Virtualized local environment

5 Manually Optimized RTOmin

When packet loss is a rare event, timeout mechanism is rarely invoked. Then RTOmin values are not critically important. When packet loss increases, intuitively the likelihood of timeout should increase as well. However with the fast retransmission mechanism in place the whole picture of interactions of timeout/retransmission/fast retransmission becomes less clear and intuitive.

The work in [16] tried to clarify relationship between RTOmin values, timeout, retransmissions, and fast retransmissions. To quantify the impact of lowered RTOmin values on fast retransmissions and SACKs, the authors there defined the notions of *spurious retransmission boundary* and *SACK interference boundary*. Various RTOmin values were tested and performance statistics were collected from the extensive testing.

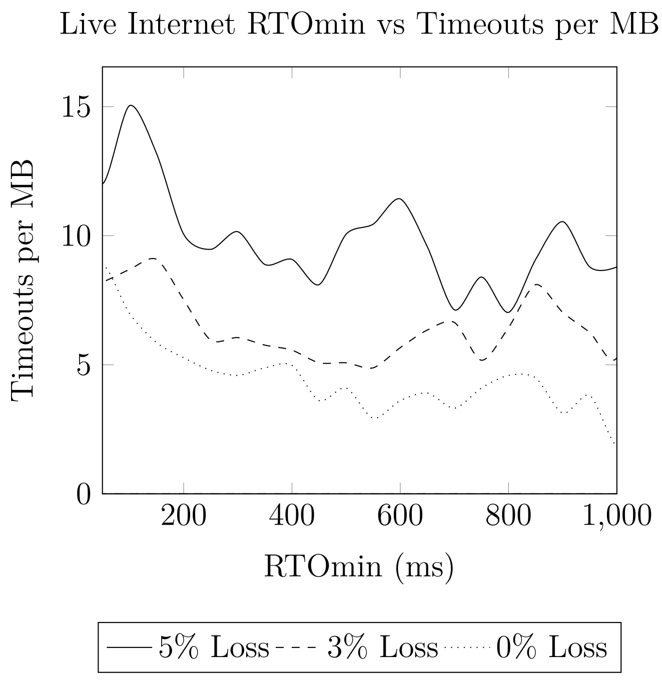

Figure 4: Live Internet RTOmin vs Timeouts per MB

Figure 4 shows the effect of RTOmin and packet loss on the number of timeout events in the live Internet test environment. It can be seen that the number of timeouts is fairly stable when RTOmin values are in the range of 250ms to 700ms. But below that range the number of timeouts begins to steadily increase. Similar results have been observed under the other two test environments.

Figure 5 illustrates the relationship between fast retransmissions and RTOmin values under live Internet test environment as well. Here the relationship is in the opposite direction than with timeouts. The higher the RTOmin the more fast retransmission events. Again similar phenomena are obtained under the other two test environments.

Based on these results and observations we hypothesize that there should be an optimization opportunity for SCTP performance by selecting an optimal or locally optimal RTOmin value for a specific network environment. This hypothesis is supported by Figures 6 (live Internet environment) and 7 (LAN local environment). In both graphs we can see that there are local maxima in goodput at a particular RTOmin. In Figure 6 we see three distinct peaks at 70ms, 550ms and 1000ms for a 0% loss connection that result in a 50% increase in throughput vs other RTOmin values. We also notice the same trend in Figure 7 where the main peaks in goodput are at 150ms and 1100ms. These peaks

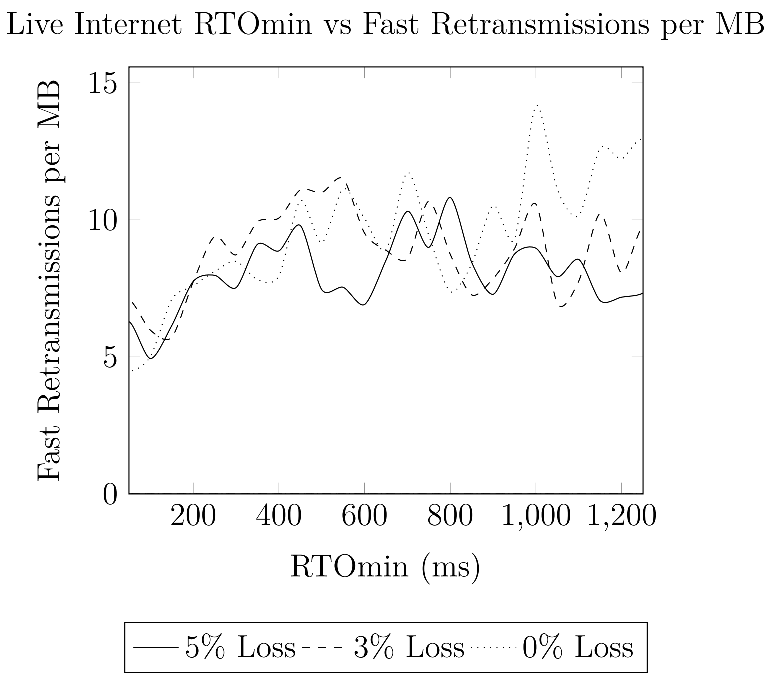

Figure 5: Live Internet RTOmin vs Fast Retransmissions per MB

are less pronounced as the packet loss increases but we can see the downward trend in goodput as the RTOmin increases. This less aggressive RTOmin begins to stifle almost all timeout events with a heavy bias towards fast retransmissions which begins lowering the goodput. With this knowledge of these particular network environments we set out to manually optimize the RTOmin for the live internet environment. Figures 8 and 9 show the result of manually selecting RTOmin values for a specific network environment.

Figure 6 and Figure 8 are results obtained from exactly the same test scenario with the same 50B chunk size. Although there is no improvement at low packet loss the manually optimized RTOmin shows higher goodput at high loss. In fact, we see the two traces begin to diverge significantly as packet loss increases. In Figure 9 the tests were conducted at 1452B chunks to see if the manually selected RTOmin at 50B chunks could benefit transfers at other chunk sizes. We see a similar trend of no improvement at low loss levels with greater improvement diverging from the SCTP standard RTOmin at higher packet loss. As discussed in [16] we believe this improvement is due to the fact that RTOmin plays a balancing role between timeout events and fast retransmission events. We can conclude that even with manually selected RTOmin values there are clearly observed signs of improvement in goodput of SCTP. This leads us to investigate and propose a dynamic and fully automatic way of harnessing RTOmin's balancing role.

6 The Dynamic RTOMin Algorithm

As was discussed in previous section RTOmin values affect both fast retransmissions and timeouts in opposite ways. In order to utilize this fact practically, we need an algorithm that can dynamically adjust RTOmin values to at least locally optimize goodput. We are also assured that RTOmin values can be significantly smaller than the standard 1000ms minimum RTO value and SCTP employing those smaller values may only incur

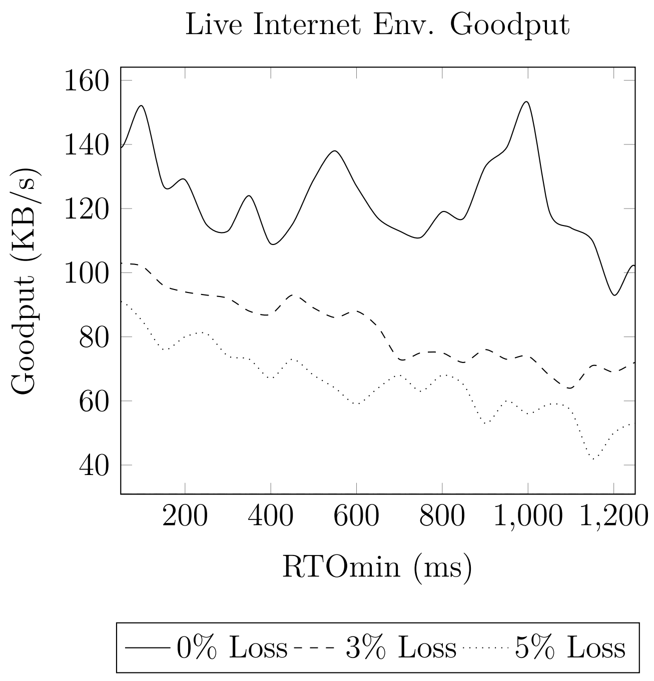

Figure 6: RTOmin vs Goodput (KB/s)

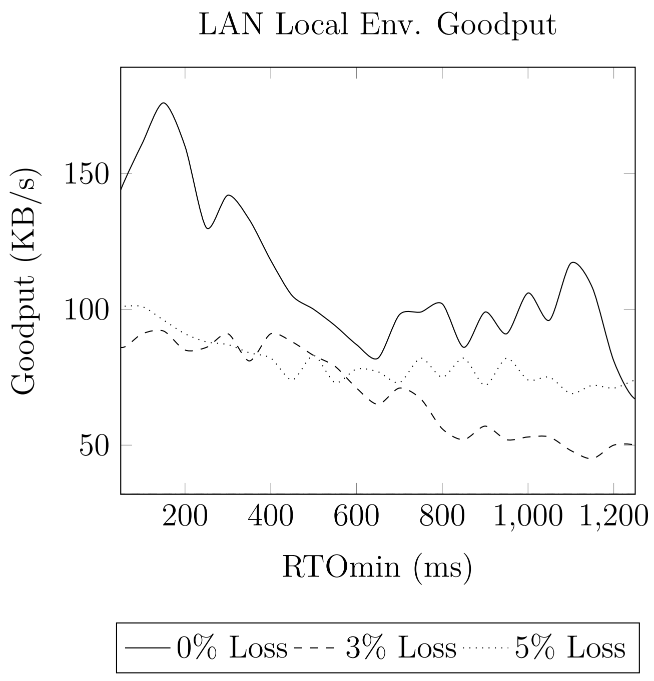

Figure 7: LAN Local Env. RTOmin vs Goodput (KB/s)

an insignificant number of timeouts or fast retransmissions. In most applications paying such a very small price is justified to gain goodput significantly.

The number of fast retransmissions and the timeout values both reflect network dynamics. The ratio of fast retransmission events to timeout events is of particular interest. Intuitively, when this ratio increases, there are fewer timeouts than fast retransmissions relatively. That in turn implies that the network environment is relatively stable and

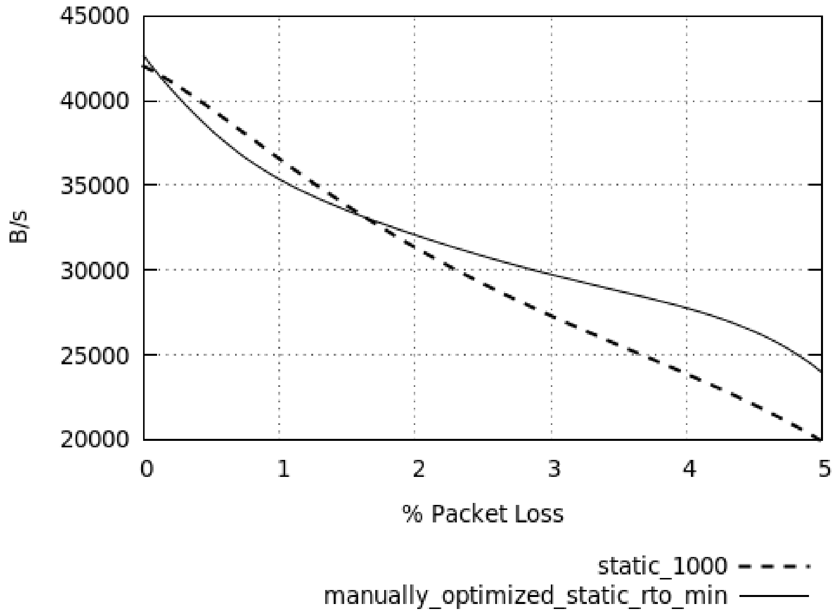

Figure 8: Throughput comparision of manual optimization vs static algorithm: 50 byte chunk size

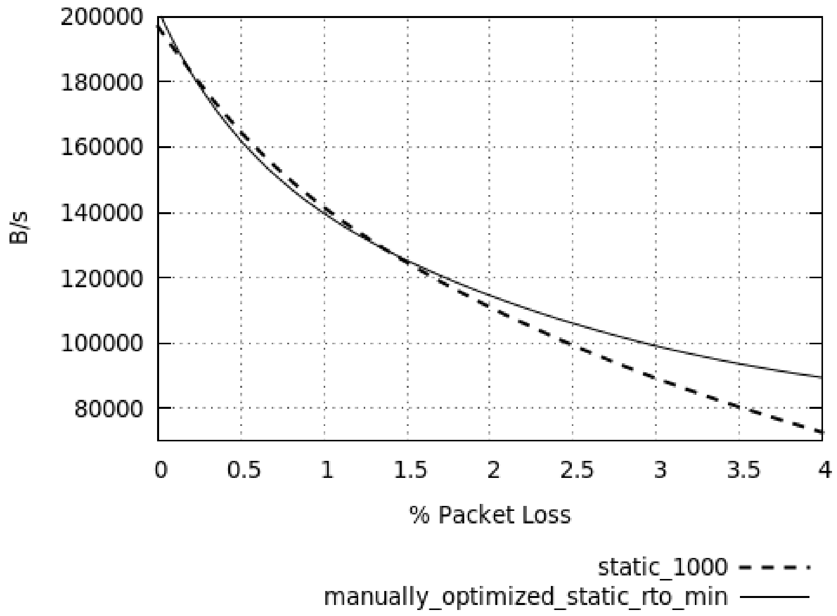

Figure 9: Throughput comparision of manual optimization vs static algorithm: 1452 byte chunk size

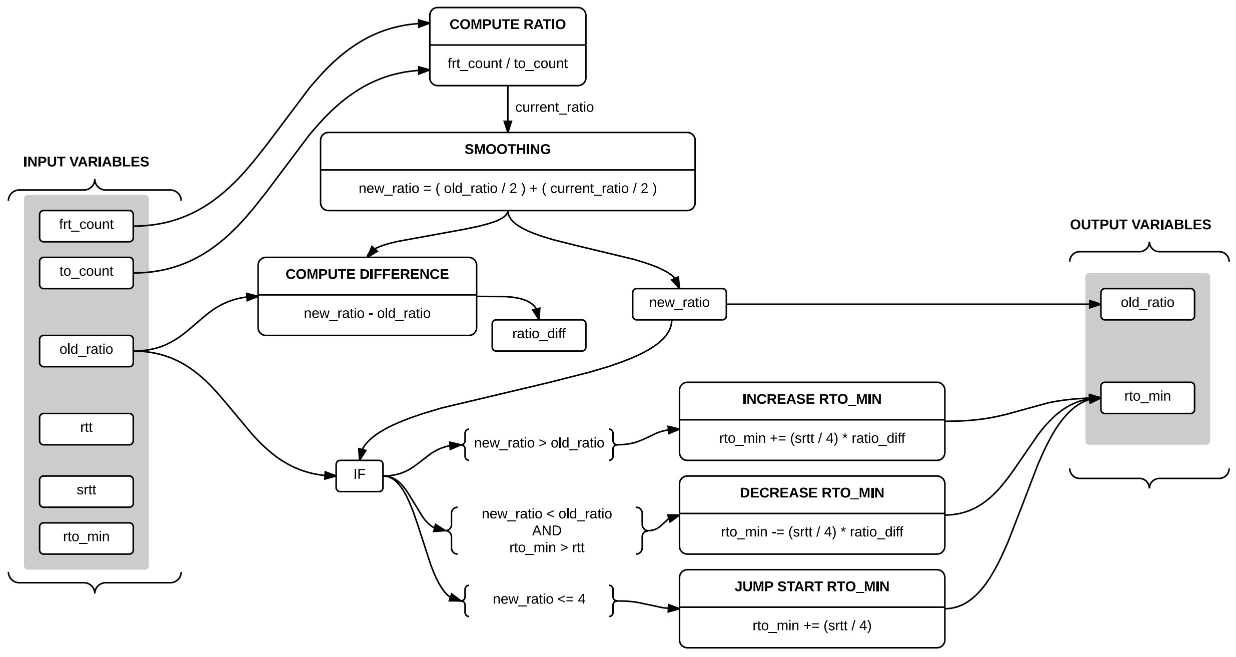

Figure 10: Dynamic RTOmin algorithm based on fast retransmission/timeout ratio

fewer packet losses occur. On the other hand, when this ratio decreases, timeouts tend to dominate fast retransmissions, which implies that the network environment is relatively unstable and more packet losses may creep up. This, combined with the fact that the RTOmin value directly affects which of the two mechanisms will trigger first, provides the core idea of our dynamic RTOmin algorithm, the main idea of which is illustrated in Figure 10.

The algorithm is implemented by modifying the SCTP stack within the Linux kernel that can run in all three test environments discussed in Section 4. In the following we briefly describe several main aspects of our implementation.

1. **New Linux kernel variables**
   (a) **frt_count** - A persistent value that is incremented whenever a fast retransmission event occurs. Its initial value is 1 and is reset at the beginning of a new association.
   (b) **to_count** - A persistent value that is incremented whenever a timeout event occurs. its initial value is 1 and is reset at the beginning of a new association.
   (c) **old_ratio** - A persistent value that keeps track of the current association's fast retransmission/timeout ratio. This value is updated at every new RTT measurement.

2. **Fast retransmission/timeout ratio calculation and smoothing -**

$$newratio = newratio - (oldratio/2) + (newratio/2) \qquad (1)$$

At every RTT measurement a ratio of the current number of fast retransmission events and timeout events is taken which is then smoothed using a similar formula

Listing 1: Ratio Algorithm Case Statement

```
//New adaptive RTOmin algorithm using FRT/TO Ratio                             1
__s32 frt_to_ratio;                                                            2
__s32 ratio_diff;                                                              3
                                                                               4
frt_to_ratio = (tp->asoc->ed_frt_count / tp->asoc->ed_to_count);               5
frt_to_ratio = tp->asoc->ed_alg_ratio -                                        6
  (tp->asoc->ed_alg_ratio >> 1) + (frt_to_ratio >> 1);                         7
ratio_diff = frt_to_ratio - tp->asoc->ed_alg_ratio;                            8
                                                                               9
if( frt_to_ratio > tp->asoc->ed_alg_ratio )                                    10
{                                                                              11
    tp->asoc->rto_min += (tp->srtt >> 2) * ratio_diff;                         12
}                                                                              13
else if( tp->asoc->rto_min > rtt &&                                            14
            frt_to_ratio < tp->asoc->ed_alg_ratio )                            15
{                                                                              16
    tp->asoc->rto_min += (tp->srtt >> 2) * ratio_diff;                         17
}                                                                              18
else if( frt_to_ratio <= msecs_to_jiffies(1) )                                 19
{                                                                              20
    tp->asoc->rto_min += (tp->srtt >> 2);                                      21
}                                                                              22
                                                                               23
tp->asoc->ed_alg_ratio = frt_to_ratio;                                         24
```

to the SRTT measurement. The formula used is shown in Equation 1 and uses a low divisor value to provide a short interval running average.

3. **Ratio Difference Calculation** - The difference between the previous "old_ratio" and the "new_ratio" is calculated in order to determine whether the ratio has decreased or increased. This value is stored in a variable called "ratio_diff" which is used in subsequent stages.

4. **Case Statement** - Depending on which direction the ratio has moved appropriate changes are made to the RTOmin value. Code snippet 1 shows the code responsible for changing the RTOmin boundary. Lines 5 shows the computation of the new ratio, lines 6-7 show the smoothing of with the currently used ratio and line 8 calculates the difference between the new ratio and the old ratio that is used in the case statement when modifying RTOmin. Lines 10-22 are dedicated to a case statement outlined below:

   (a) **If new_ratio > old_ratio** Implies that we need to continue favoring fast retransmissions over timeouts hence we increase RTOmin by (SRTT / 4) * ratio_diff which ensures we increase proportionately by the amount the ratio has increased.

   (b) **If new_ratio < old_ratio** Implies that we have encountered link loss which, due to its unlikely nature, means that the timeout mechanism might be the favorable alternative. Thus, we reduce RTOmin by (SRTT/4) * ratio_diff

(c) **If new_ratio < 4** If the new_ratio is too low then we assume we are at the beginning of an SCTP transfer and we jumpstart the Ratio Algorithm by increasing RTOmin by (SRTT / 4) in order to begin increasing the number of fast retransmissions over timeouts.

Intuitively, by jointly considering fast retransmission events and timeout events, congestion and network loss can be differentiated and intelligently managed. This is the basis of the dynamic RTOmin algorithm.

In current SCTP implementations and modern networks, the fast retransmission mechanism is much more likely to respond before a timeout occurs. In a low loss network, timeout events are rare and the likelihood of a timeout increases with packet loss, which is uncommon. Figures 11 and 12 describe this phenomenon very clearly. Figure 11 shows experimental measurements of timeout events (TO) over a variety of network conditions, including increasing packet loss. Figure 12 examines fast retransmissions (FRT) over the same conditions. In the figures, TO and FRT measurements are normalized by throughput for each set of test iterations, and smoothed to show the general trend. The performance of the conventional "static RTOmin algorithm" (a fixed 1000 msec threshold) is shown with a dashed line. The performance of our modified approach, or the "dynamic RTOmin algorithm" is shown with a solid line. The dynamic RTOmin algorithm is described in detail in later sections.

It is important to understand the relationship between FRT and TO. In examining the performance of the "static RTOmin" approach in Figures 11 and 12 (dashed line), the inversely related trends of TO and FRT are very apparent. For example, around packet loss ratios of 0.5%, FRT events are frequent (around 60 FRT/MB), and TO events are infrequent (around 2 TO/MB). As packet loss increases, the rate of TO increases (10 TO/MB), while FRT events are fewer (10 FRT/MB). These trends match conventional wisdom, and validate the operation of the well-known retransmission mechanisms described previously. In a "fast, clean" link, packet loss is sparse and fast retransmissions are more likely than timeouts. As packet loss increases and network conditions proceed towards link failure, timeouts become more prevalent and fast retransmissions diminish.

Figure 13 takes a look at the relationship between FRT and TO from another angle. It illustrates the FRT/TO ratio for both the static and dynamic RTOmin algorithm. Once again the static algorithm trends towards the dynamic algorithm at very high losses but we see that under 3% packet loss the ratio is as high as 500. This is a disproportionate amount of FRT that should not be happening for best goodput. In comparison the dynamic RTOmin algorithm's curve shows a small increase at 1% loss and then remains very steady for the remainder of the tests. This allows the TO mechanism to step in more often and act appropriately aggressively in retransmitting packets.

The inverse nature of FRT and TO behavior under varying packet loss provides the basic concept of the dynamic RTOmin algorithm, which is driven by the ratio FRT/TO. Figure 10 provides a pseudocode illustration of the dynamic RTOmin algorithm. At every RTT measurement, we compute the ratio of the fast retransmission events and timeout events (FRT/TO). This sequence of values is then filtered using a single-pole, infinite-impulse-response filter. The filtered difference between the previous and current values of the FRT/TO ratio provides an effective measure of network context. If the FRT/TO

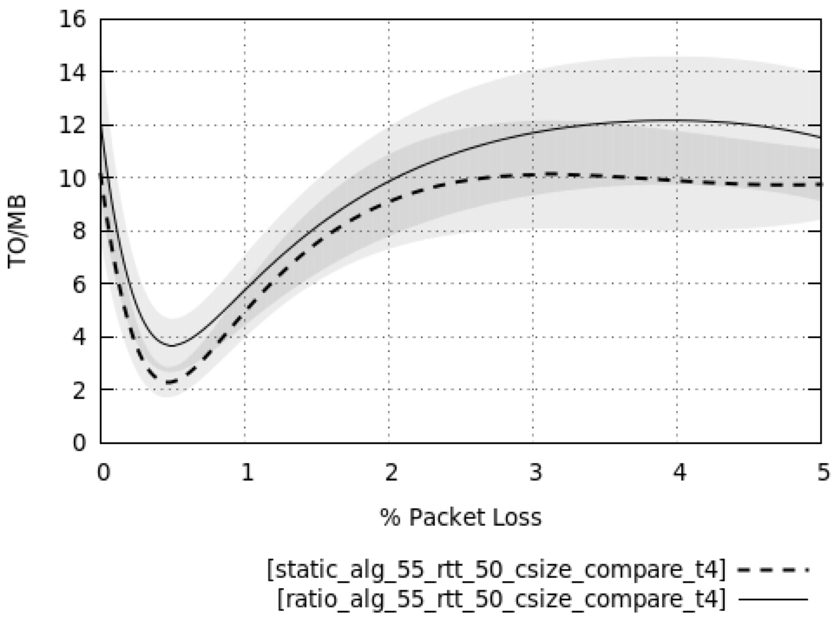

Figure 11: Timeout events: static vs. dynamic RTOmin

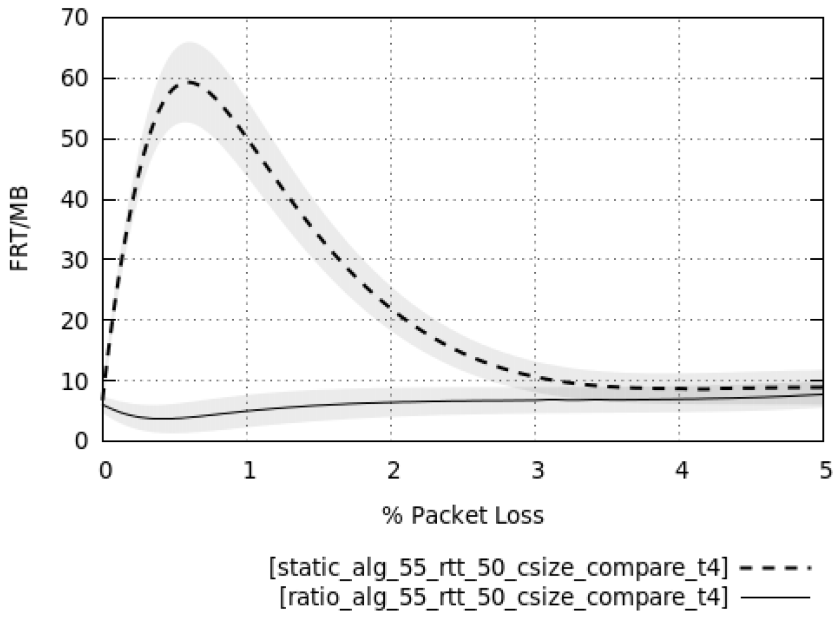

Figure 12: Fast retransmission events: static vs. dynamic RTOmin

ratio is *decreasing*, the subsequent RTOmin bounding value is lowered by a dynamic scaling factor. Conversely, if the FRT/TO ratio is *increasing*, the RTOmin boundary is raised by a similar value. The scaling factor for the RTOmin adjustment takes into account the first-order rate of change of the FRT/TO ratio as well as the current average RTT. In essence, the dynamic RTOmin algorithm attempts to find the first maxima of the curve described by the FRT/TO ratio. By increasing RTOmin until the FRT/TO ratio begins to decrease, the system leverages an initial aggressive RTOmin value which

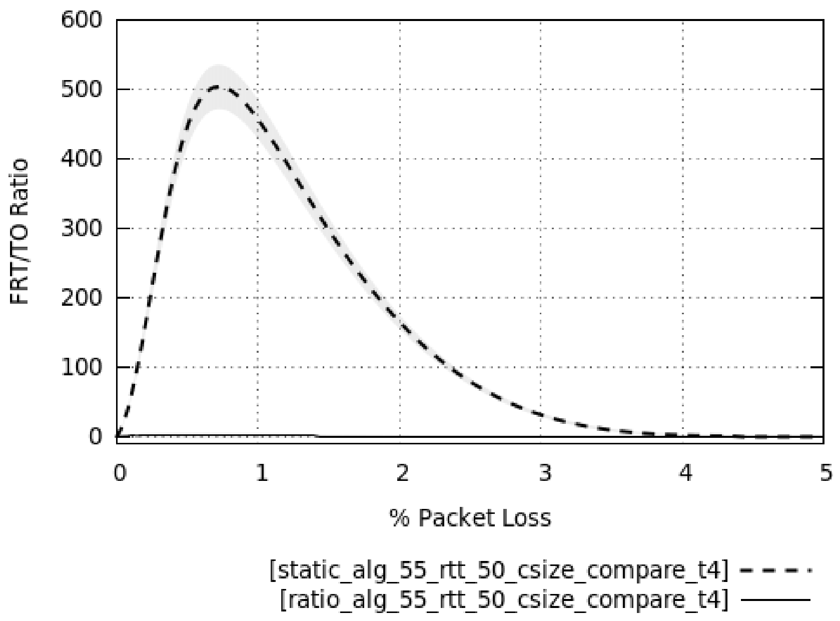

Figure 13: Fast retransmission/timeout ratio: static vs dynamic RTOmin

provides quick detection of loss on the link. If the link has low loss, then the RTOmin value increases to avoid spurious retransmissions.

Since RTOmin bounding is applied to the *output* of Jacobson's algorithm, the value of RTOmin directly affects which of the two mechanisms (FRT or TO) will occur first. Thus, the dynamic RTOmin approach uses network conditions to *dynamically adjust the context* in which Jacobson's algorithm operates. As a result, in network configurations where a timeout may improve throughput, the RTOmin boundary is pre-adjusted to the likelihood of a timeout, rather than "stuck" at a fixed value of 1000 msec, completely ignoring network context. Conversely, in network configurations where the average RTT is very large, the dynamic value of RTOmin "floats" above the previously fixed 1000 msec threshold, preventing a large number of consecutive timeouts and or fast retransmission in slow networks.

By working in conjunction with existing retransmission mechanisms, the dynamic RTOmin approach uses network variability to intelligently adjust the operating context for Jacobson's algorithm and other retransmission mechanisms. The net result is that Jacobson's algorithm is allowed to participate more actively in link management. The approach works particularly well for "fast, clean" links.

7 Experimental Results

The dynamic RTOmin algorithm has been tested in all the three test environments described in 4. In the live Internet environment client/server hosts communicate through the Internet, and between different network providers. The average RTT encountered during our testing was observed to be between 50 msec and 150 msec, well below the 1000 msec of "static RTOmin". For completeness, secondary test scenarios used a similar configuration in the LAN local environment as well as in the virtualized local environment to emulate datacenter networking trends. The LAN local environment displayed average RTT values on the order of 5-10 msec, while average RTT in the virtualized local environment was under 1 msec. Only Internet-based testing is presented here.

In all cases, we use client hosts running 32 bit Debian Linux and server hosts running 32 bit CentOS 6. All hosts use Linux kernel version 2.6.32, and communicate via an intervening routed network. The server hosts have unmodified SCTP protocol implementations, and capture data from the transmission tests. The dynamic RTOmin algorithm is implemented as modified SCTP modules in the Linux kernel, and is only required on the client-side. During a test, client systems swap between unmodified and modified SCTP Kernel modules for different iterations to directly compare the performance of the new algorithm with the existing Linux implementation [25]. Network conditions such as loss and delay were manipulated at the client NICs using the Linux Network Emulator [24].

In each test sequence, the servers wait for connections from clients, then log relevant parameters. The clients initialize connections and log relevant parameters while transferring data. For subsequent test iterations, client systems swap between modified and unmodified kernel modules. In all cases, a single "test iteration" involves transfer of SCTP chunks of a specified size via links with packet losses from 0% to 5%. For each test iteration, several thousand transfers of chunks were attempted, producing statistically viable data. The shaded areas in Figures 11, 12, and Figure 16 show 95% confidence intervals for each test. For the purposes of this paper we focus on the transfer of 50-byte chunks, but other chunk sizes provide similar results.

7.1 Performance with Packet Loss

Figure 14 shows recorded test parameters for the dynamic RTOmin algorithm from the beginning of a single transfer in a very low loss environment. This trace was taken from a link with an average RTT of 55 msec and a transfer using 50-byte chunks. The steady increase of RTOmin can be seen as the transfer progresses. For this particular transfer the RTOmin value begins settling around 500 msec. This value is well above the delayed SACK time-out of 200 msec which ensures that there will be no interference, and well below the conventional "static RTOmin" value of 1000 msec.

Figure 15 shows recorded test parameters for the dynamic RTOmin algorithm from the beginning of a single transfer in a high loss link. From 0 to 1.75 seconds, the initial rise of RTOmin is very clear. Starting at 2 seconds, several drops indicate that the FRT/TO ratio is decreasing, so the dynamic algorithm lowers the RTOmin boundary in response. After 4 seconds, the RTOmin value settles slightly below 200 msec. Since this is a high-loss link, the fact that many packets are being lost lessens the impact on the number of delayed SACKs. Note also that the dynamic RTOmin algorithm proactively *reduces* RTOmin in the lossy link, where fast retransmissions are less effective, in preparation for timeout events. Conversely, RTOmin is *raised* in the low-loss link to allow for more efficient fast retransmission events and provide additional operating context for Jacobson's algorithm.

7.2 Goodput Performance

To emphasize the effect of the dynamic RTOmin algorithm, we measure the *goodput* of the transmission tests along with the recorded retransmission parameters. Figure 16 shows the goodput of the conventional "static RTOmin" of 1000 msec (dashed line) versus the dynamic RTOmin algorithm (solid line). As in previous cases, testing encompassed

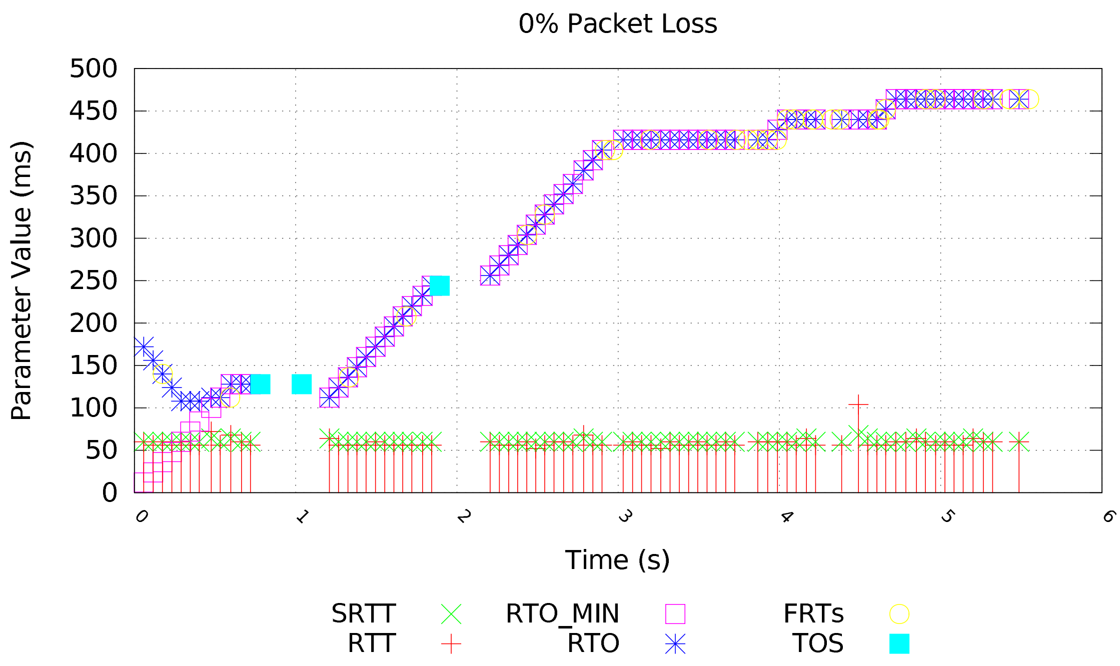

Figure 14: Dynamic RTOmin with packet loss of 0%. After an initial onset period (0-2 sec), the RTOmin boundary “settles” just below 500 msec.

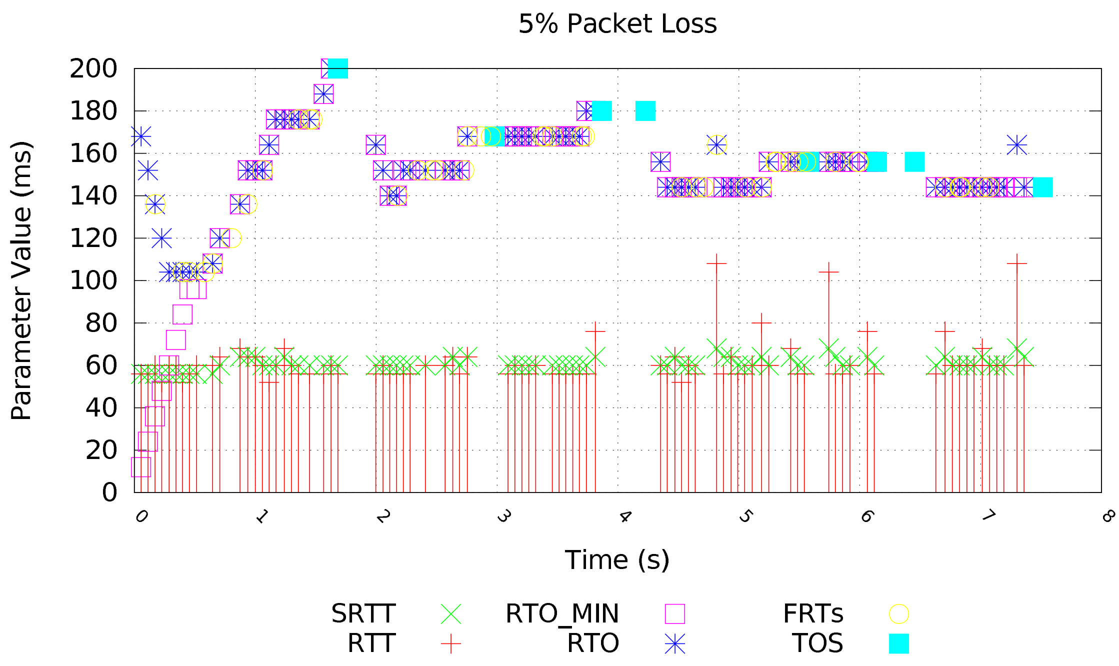

Figure 15: Dynamic RTOmin with packet loss of 5%. The RTOmin boundary “settles” near 200 msec in a slightly more aggressive approach to retransmission and link failure detection.

links with packet losses from 0% to 5%. For each packet-loss value, several thousand transfers of 50-byte chunks were attempted. The figure clearly shows that the dynamic RTOmin algorithm exhibits the same general behavior as the static RTOmin case, but it does so at a uniformly higher goodput, with improvement from roughly 50k to 70k B/s, or about 40% beginning at 2% packet loss. Similarly, in the "clean network" region around 0.5% packet loss, the dynamic approach improves goodput by about 6% from 160k to 170k B/s.

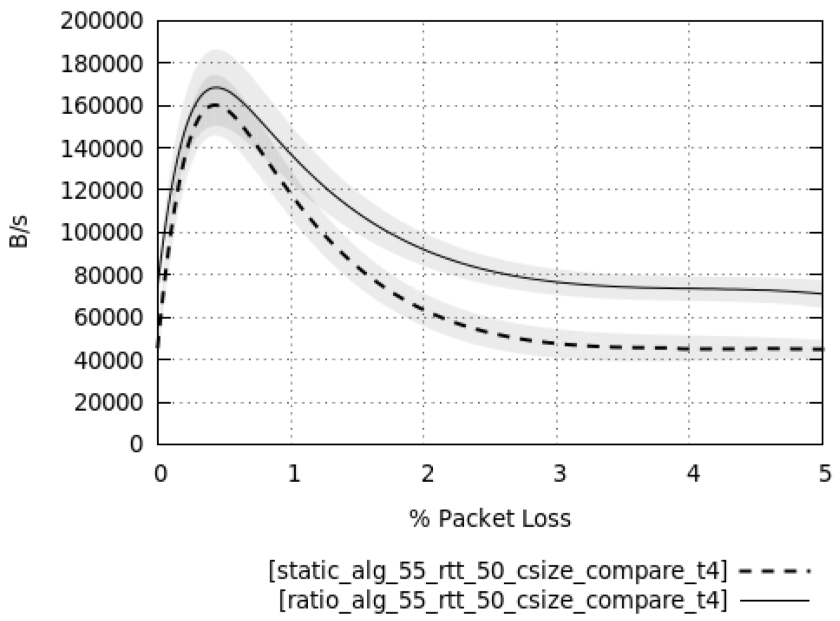

Figure 16: Goodput: static vs. dynamic RTOmin

This behavior is further confirmed by the TO and FRT curves of Figures 11 and 12, which contain data from the same test environment. In Figure 11, note that the static and dynamic RTOmin cases exhibit similar performance. Further, the dynamic RTOmin approach exhibits a small but consistent increase of TO events over all packet loss scenarios. Thus, the overall nature of the transfers have not changed, but *Jacobson's algorithm is engaged more often* and producing only slightly more TO events. Note from Figure 11 that the dynamic RTOmin (solid line) approach produces roughly one additional TO/MB compared to static approach. In Figure 12 the dynamic RTOmin algorithm (solid line) maintains a consistent number of fast retransmission events regardless of the packet loss, while the static RTOmin approach (dashed line) spikes at 0.5% packet loss with 60 FRT/MB, then asymptotically approaches the operating point of the dynamic algorithm at 10 FRT/MB. This behavior shows that the dynamic RTOmin approach does not interfere with the nature of the session. These in-situ observations of FRT and TO events, along with goodput effectiveness support the assertion that the dynamic RTOmin algorithm balances effectively between fast retransmissions and timeouts to improve the throughput of SCTP sessions.

8 Conclusions

This paper presents a dynamic RTOmin algorithm that dynamically manages the RTOmin boundary to enhance throughput in SCTP. The current RTOmin is usually set to such a high value (1000 msec) that the timeout mechanism is hardly active until the entire link is down. This leaves the fast retransmission mechanism in charge of both detecting lost packets and detecting network congestion, which is not optimal [21]. The dynamic approach uses a ratio of fast retransmission events and timeout events to effectively differentiate between congestion and network loss. This allows an initial aggressive RTO which provides quick detection of loss on the link. If the link has low loss then the RTOmin value increases to avoid spurious retransmissions.

Our experiments have showed that that RTOmin values can be significantly smaller than the standard 1000ms minimum RTO value and SCTP employing those smaller values may only incur an insignificant number of timeouts or fast retransmissions. The proposed dynamic RTOmin algorithm adjusts RTOmin values according to the change of network dynamics and can provide RTOmin values that are locally or globally optimal for goodput.

The dynamic algorithm has been implemented and tested in real network environments. The extensive testing in these environments shows that the algorithm significantly enhances goodput versus the current SCTP timeout scheme without interfering with existing, well-known techniques. We believe that our research presented here is a solid step forward toward a new timeout scheme for stream protocols (SCTP and TCP) that is both efficient and stable.

We believe that our research is an important step toward a uniform and dynamic timeout control of stream protocols, including SCTP and TCP. The experimental nature of the research merits special attention. The algorithm has been tested extensively under real network environments.

9 Future Work

In future work, we will investigate the effectiveness of the dynamic RTOmin algorithm in a number of other environments and with alternate transport protocols, including embedded systems for "Internet of Things" applications as well as virtualized environments for datacenter applications. Additionally, we continue to experiment with alternative approaches for retransmission timeout estimation as well as optimization of current algorithms.

References

[1] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol, RFC 2960," October 2000. [Online]. Available: http://dx.doi.org/10.17487/RFC2960

[2] R. Stewart, "Stream Control Transmission Protocol, RFC 4960," September 2007. [Online]. Available: http://dx.doi.org/10.17487/RFC4960

[3] P. Natarajan, F. Baker, P. Amer, and J. Leighton, "SCTP: What, why, and how," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 81–85, Sep.-Oct. 2009. [Online].

Available: http://dx.doi.org/10.1109/MIC.2009.114

[4] V. Jacobson, "Congestion avoidance and control," in *Proc. Comput. Commun. Review*, ser. SIGCOMM'88. New York, NY: ACM, Aug. 1988, pp. 314–329. [Online]. Available: http://dx.doi.org/10.1145/52324.52356

[5] V. Jacobson and R. Braden, "TCP extensions for long-delay paths, RFC 1072," Oct. 1988. [Online]. Available: http://dx.doi.org/10.17487/RFC1072

[6] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance, RFC 1323," May 1992. [Online]. Available: http://dx.doi.org/10.17487/RFC1323

[7] S. McClellan and W. Peng, "Estimating retransmission timeouts in IP-based transport protocols," in *Proc. ICDT 2013*, Apr. 2013, pp. 26–31, ISBN: 978-1-61208-262-2.

[8] R. Ludwig and K. Sklower, "The Eifel retransmission timer," in *Proc. ACM Comput. Commun. Review (SIGCOMM'00)*, July 2000, pp. 17–27, ISSN: 0146-4833. [Online]. Available: http://dx.doi.org/10.1145/382179.383014

[9] S. McClellan and W. Peng, "Improving retransmission performance of IP-based transport protocols," *Int. J. Adv. Telecomm.*, vol. 6, no. 3 & 4, pp. 123–131, 2013, ISSN: 1942-2601. [Online]. Available: http://www.iariajournals.org/telecommunications/

[10] S. Hamroui, J. Lloret, P. Lorenz, and M. Lalam, "TCP performance in mobile ad-hoc networks," *Network Protocols and Algorithms*, vol. 5, no. 4, pp. 117–142, Dec. 2013. [Online]. Available: http://dx.doi.org/10.5296/npa.v5i4.4773

[11] J. Pedersen, "Evaluation of SCTP retransmission delays," Master's thesis, University of Oslo, May 2006. [Online]. Available: http://heim.ifi.uio.no/~paalh/students/JonPedersen

[12] A. Petlund, P. Beskow, J. Pedersen, E. S. Paaby, C. Griwodz, and P. Halvorsen, "Improving SCTP retransmission delays for time-dependent thin streams," *Multimedia Tools Applied*, no. 45, pp. 33–60, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11042-009-0286-8

[13] J. Eklund, A. Brunstrom, and K.-J. Grinnemo, "On the relation between SACK delay and SCTP failover performance for different traffic distributions," in *Int. Conf. on Adv. Comm. Tech. (ICACT)*, 2008. [Online]. Available: http://dx.doi.org/10.1109/BROADNETS.2008.4769145

[14] I. Psaras and V. Tsaoussidis, "The TCP minimum RTO revisited," in *Proc. 6th Int. IFIP-TC6 Conf.*, 2007, pp. 981–991. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72606-7_84

[15] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi, "Proportional rate reduction for TCP," in *Proc. 11th ACM SIGCOMM Conf. on Internet Measurement*, Berlin, Germany, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1145/2068816.2068832

[16] E. Gonzalez, S. McClellan, and W. Peng, "RTOmin as a balancing parameter between fast retransmissions and timeouts within stream control transmission protocol (SCTP)," in *Proc. IEEE 2014 International Conference on Systems and Informatics (ICSAI)*, Nov. 2014, pp. 687–691. [Online]. Available: http://dx.doi.org/10.1109/ICSAI.2014.7009373

[17] M. Chowdhury and I. Jony, "An experimental study of SCTP in NS2," *Network Protocols and Algorithms*, vol. 6, no. 3, pp. 103–118, Aug. 2014. [Online]. Available: http://dx.doi.org/10.5296/npa.v6i3.5535

[18] I. Najm, M. Ismail, J. Lloret, K. Ghafoor, B. Zaidan, J. Lloret, and A. Rahem, "Improvement of SCTP congestion control in the LTE-A network," *Journal of Network and Computer Applications*, Sept. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2015.09.003

[19] T. Dreibholz, E. Rathgeb, I. Rungeler, R. Seggelmann, M. Tuxen, and R. Stewart, "Stream Control Transmission Protocol: past, current, and future standardization activities," *Communications Magazine, IEEE*, vol. 49, no. 4, pp. 82–88, Apr 2011. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2011.5741151

[20] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options, RFC 2018," October 1996. [Online]. Available: http://dx.doi.org/10.17487/RFC2018

[21] R. R. Stewart and Q. Xie, *Stream Control Transmission Protocol (SCTP): a reference guide*. Addison Wesley, 2002, ISBN-13: 078-5342721867.

[22] E. G. Guerra, "Reducing SCTP's time-to-complete by manipulation of its retransmission time out minimum," Master's thesis, Texas State University - Department of Computer Science, August 2014. [Online]. Available: https://digital.library.txstate.edu/handle/10877/5278

[23] L. Budzisz, J. Garcia, A. Brunstrom, and R. Ferrus, "A taxonomy and survey of SCTP research," *ACM Computing Surveys*, vol. 44, no. 4, pp. 1–36, August 2012. [Online]. Available: http://dx.doi.org/10.1145/2333112.2333113

[24] *netem*, The Linux Foundation, 2009. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[25] L. M. H. Yarroll and K. Knutson, *Linux Kernel SCTP : the third transport*, 2001. [Online]. Available: http://old.lwn.net/2001/features/OLS/pdf/pdf/sctp.pdf

Copyright Disclaimer